specRTL: A Language for GCC Machine Descriptions

Uday Khedker

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

April 2011

Outline

- Introduction and motivation
- Introduction to specRTL
- Conclusions and Future Work

Outline

- Introduction and motivation
- Introduction to specRTL
- Conclusions and Future Work
- *Disclaimer: Preliminary work that is still evolving*

Part 1

Introduction

Compilation Models

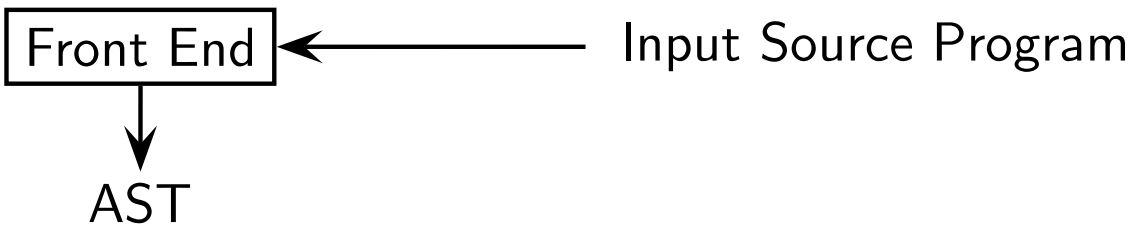
Aho Ullman Model

Davidson Fraser Model

Compilation Models

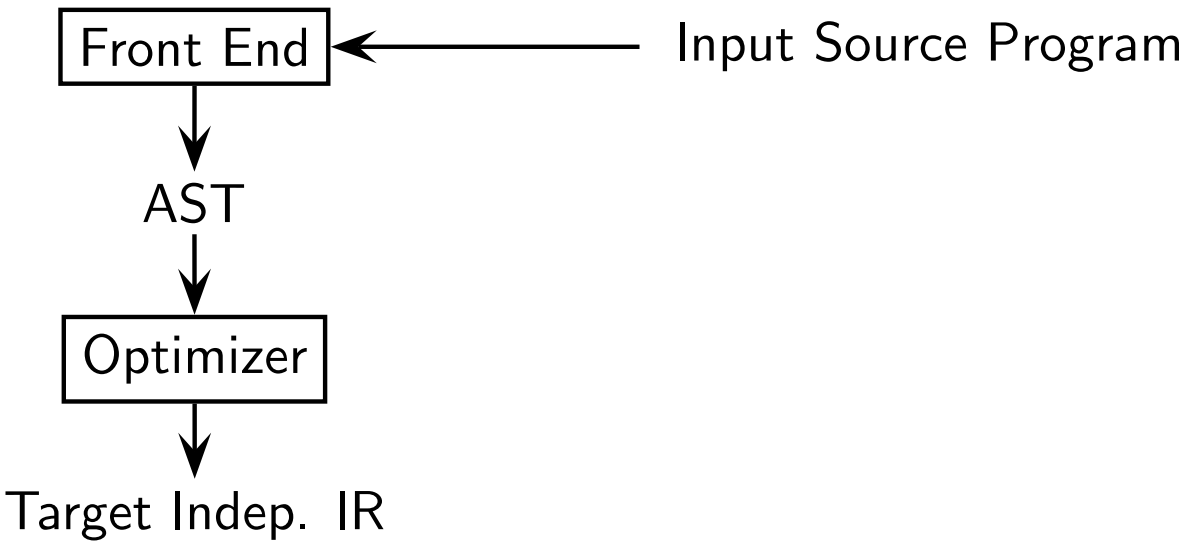
Aho Ullman Model

Davidson Fraser Model



Compilation Models

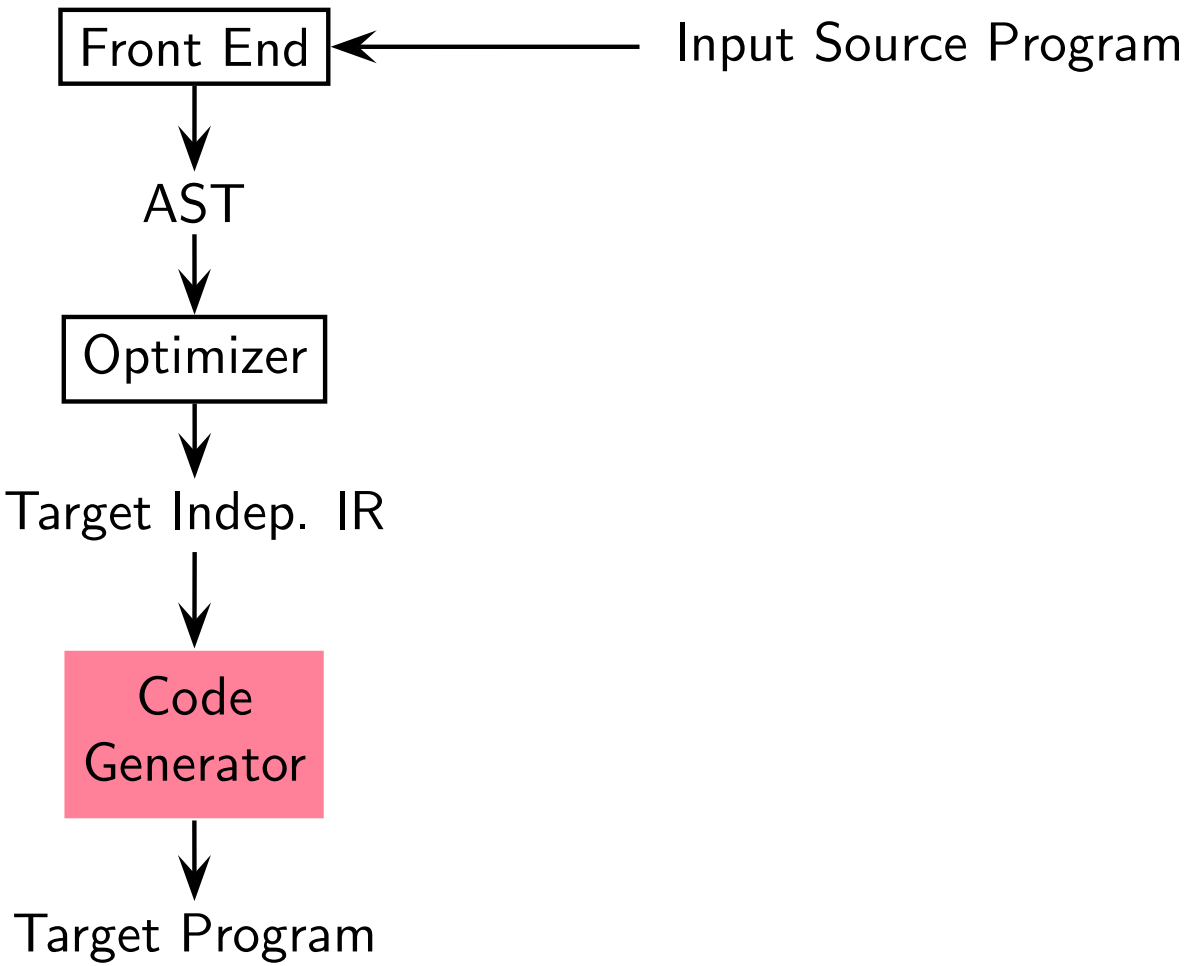
Aho Ullman Model



Davidson Fraser Model

Compilation Models

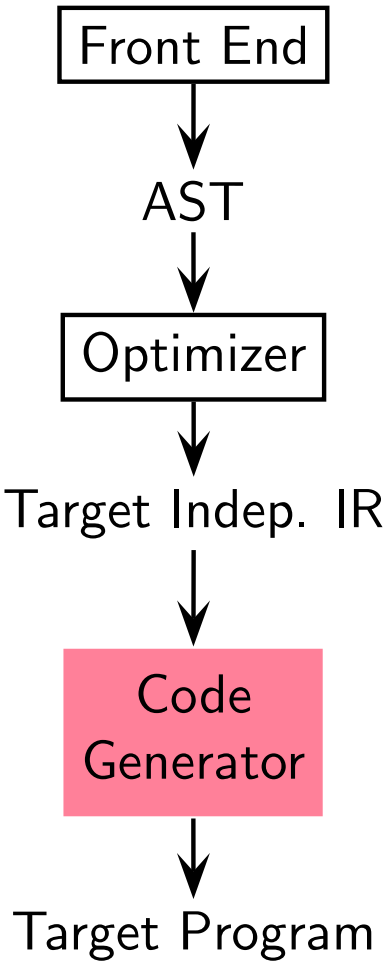
Aho Ullman Model



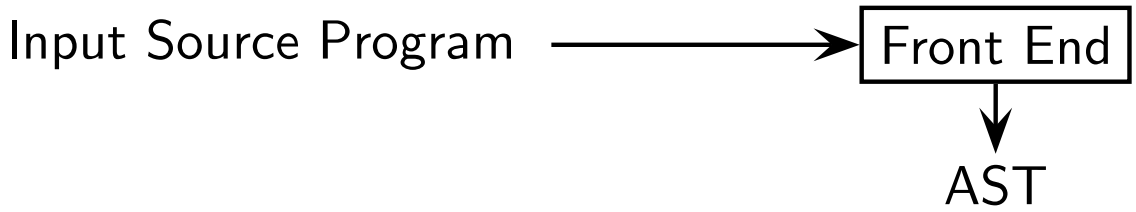
Davidson Fraser Model

Compilation Models

Aho Ullman Model

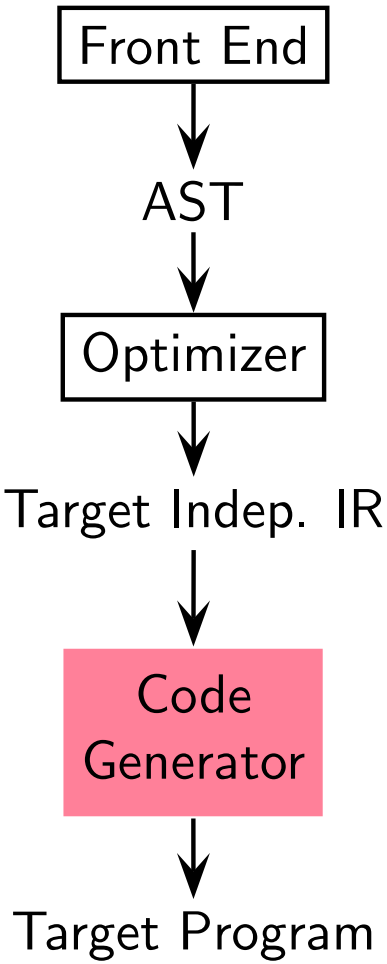


Davidson Fraser Model

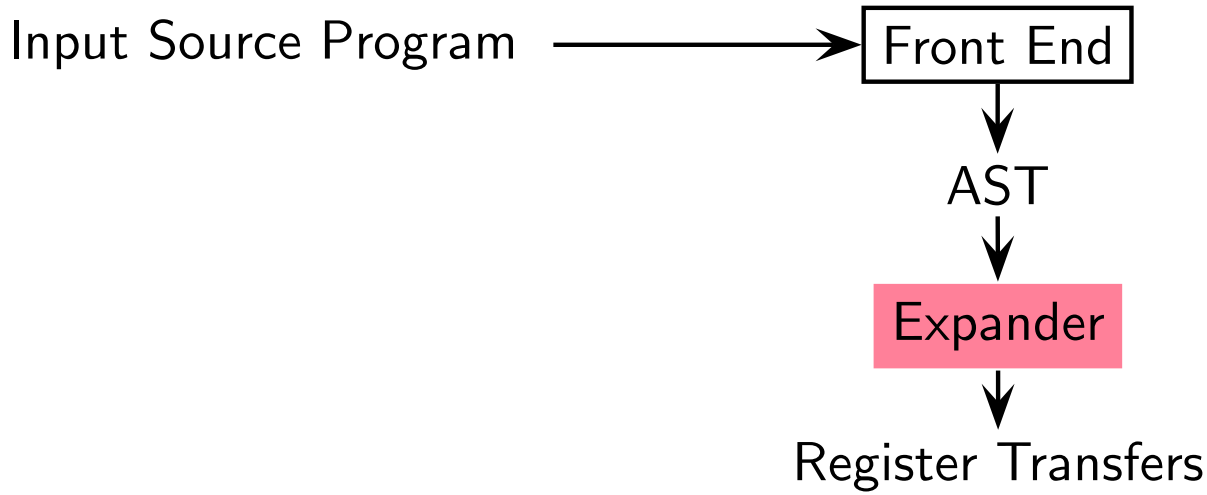


Compilation Models

Aho Ullman Model

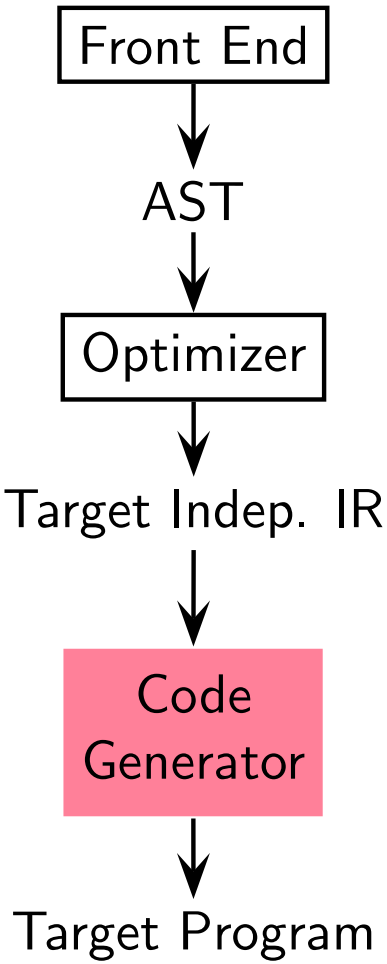


Davidson Fraser Model

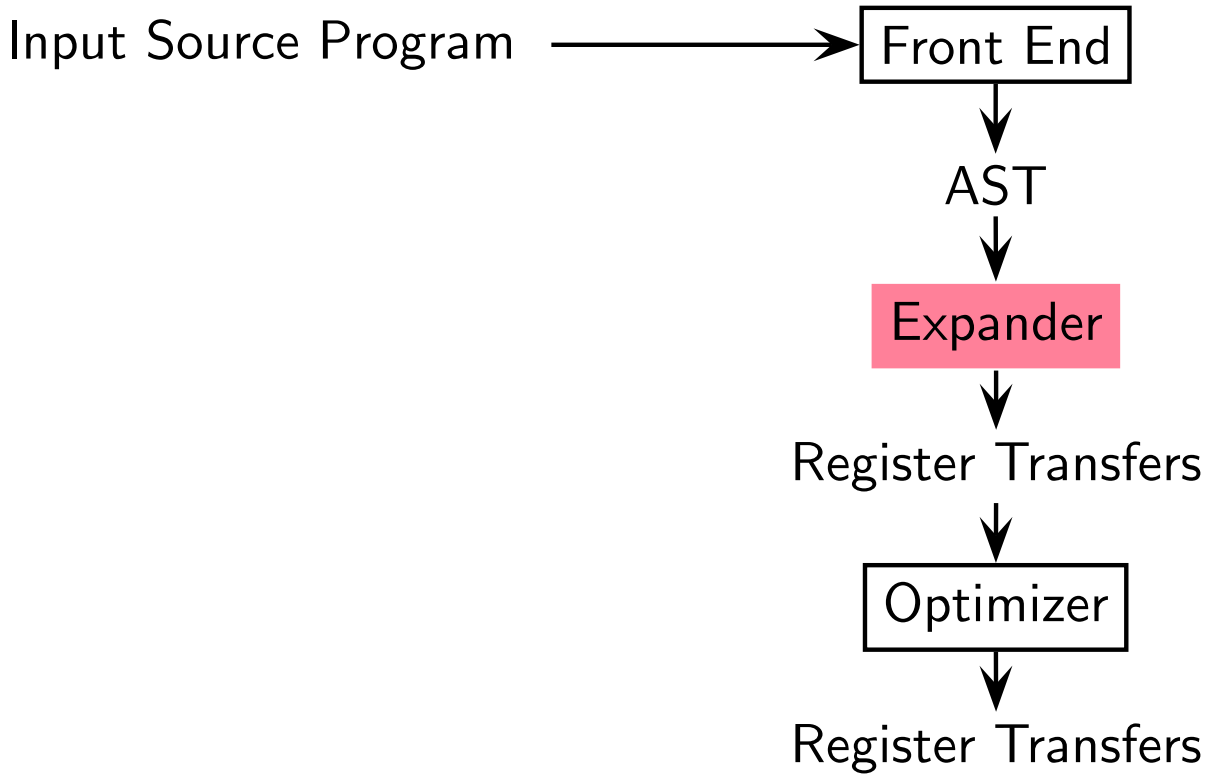


Compilation Models

Aho Ullman Model

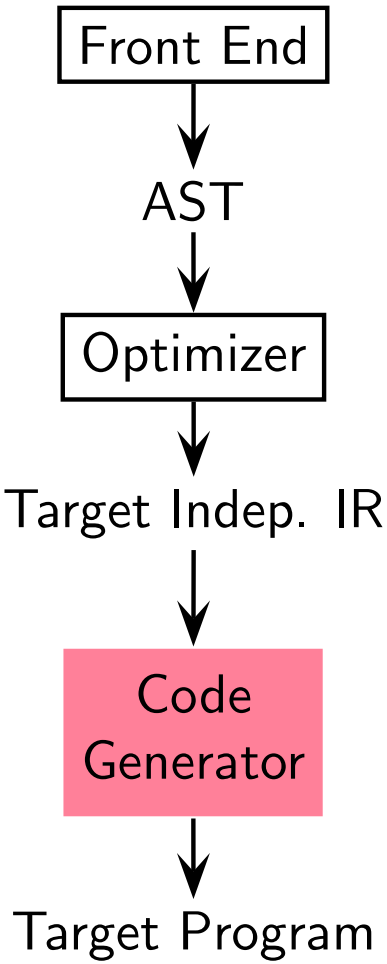


Davidson Fraser Model

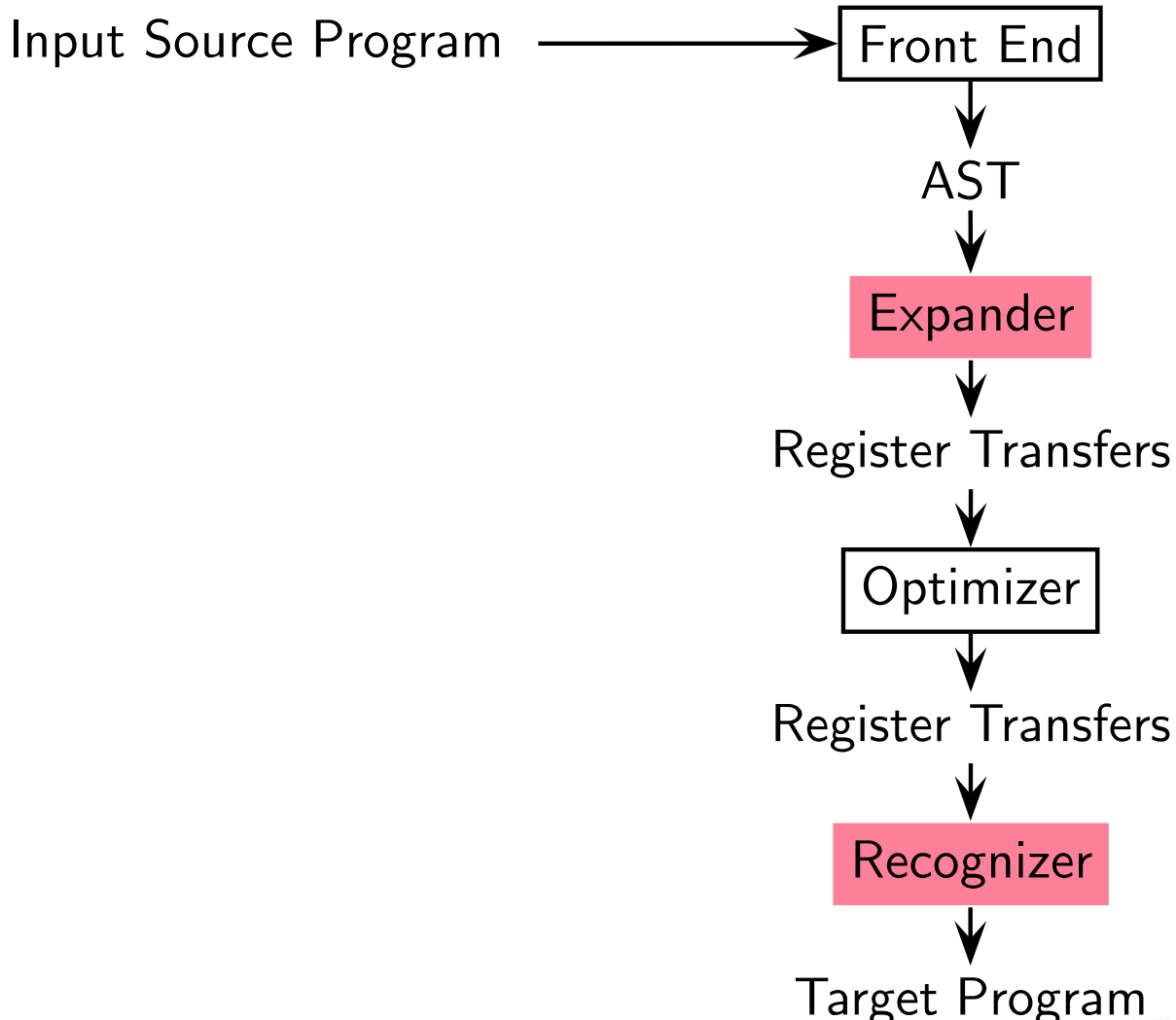


Compilation Models

Aho Ullman Model

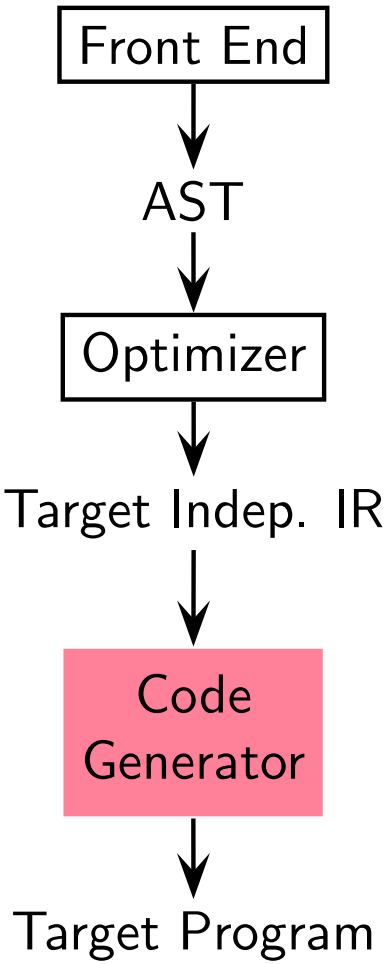


Davidson Fraser Model



Compilation Models

Aho Ullman Model



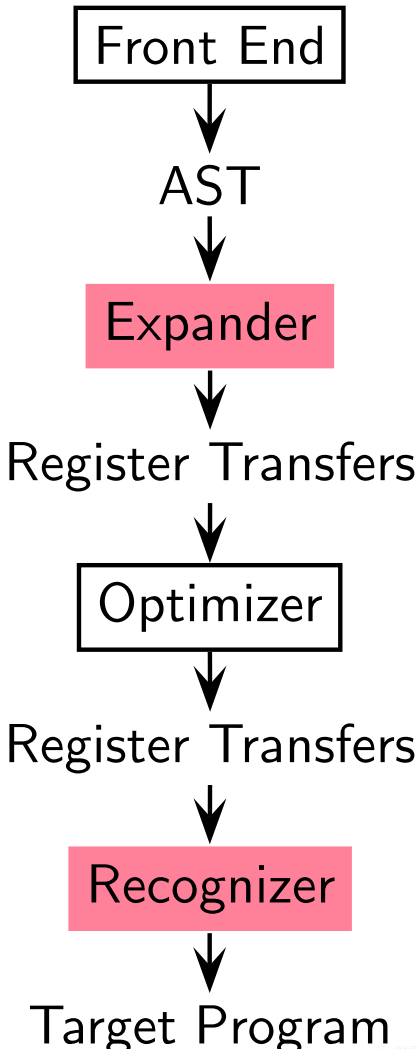
Aho Ullman: Instruction selection

- over optimized IR using
- cost based tree pattern matching

Davidson Fraser: Instruction selection

- over AST using
- structural tree pattern matching
- naive code which is
  - target dependent, and is
  - optimized subsequently

Davidson Fraser Model



Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Optimization</td><td></td><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td></td></tr>
<tr><td>Optimization</td><td></td><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structual tree pattern matching</td></tr>
<tr><td>Optimization</td><td></td><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structual tree pattern matching</td></tr>
<tr><td>Optimization</td><td>Machine independent</td><td></td></tr>
</table>

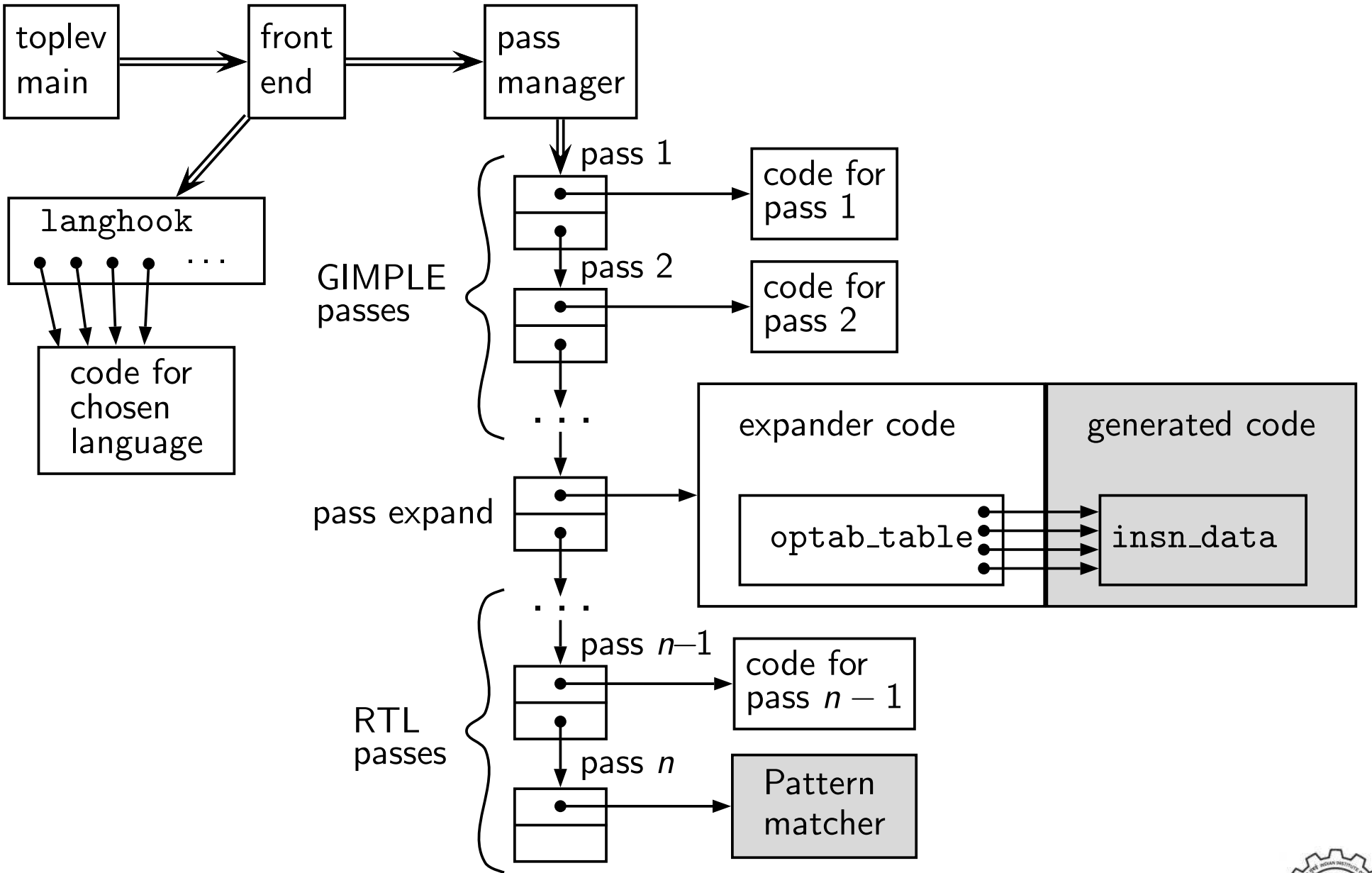
Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structual tree pattern matching</td></tr>
<tr><td rowspan="2">Optimization</td><td rowspan="2">Machine independent</td><td>Machine dependent</td></tr>
<tr><td></td></tr>
</table>

Retargetability in Aho Ullman and Davidson Fraser Models

<table>
<tr><th></th><th>Aho Ullman Model</th><th>Davisdon Fraser Model</th></tr>
<tr><td rowspan="2">Instruction Selection</td><td colspan="2">• Machine independent IR is expressed in the form of trees
• Machine instructions are described in the form of trees
• Trees in the IR are “covered” using the instruction trees</td></tr>
<tr><td>Cost based tree pattern matching</td><td>Structual tree pattern matching</td></tr>
<tr><td rowspan="2">Optimization</td><td rowspan="2">Machine independent</td><td>Machine dependent</td></tr>
<tr><td>Key Insight: Register transfers are target specific but their form is target independent</td></tr>
</table>

GCC's Adaptation of Davidson Fraser Model



Comparing Code Generators in Davidson Fraser Model

		GCC	Zephyr/VPO	Quick C--
Expander	Transformation Trees (*TT*)	RTL templates	RTL templates	Expansion tiles
	Nature of *TT*	Target dependent	Target dependent	Target independent
	Fixing shapes of *TT*	MD writing	MD writing	Framework design
Recognizer	*TT*→*Inst* method	Pattern matching using finite automaton	LR parsing (Yacc based)	Pattern matching
	TT→*Inst* mapping	Fixed manually	Discovered automatically	Discovered automatically
	Time of devising *TT*→*Inst* mapping	MD writing	Compilation	Compiler construction

Part 2

Motivation

The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc

- Adhoc design decisions

The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc

- Adhoc design decisions

The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
  - Mode and code iterator mechanisms are insufficient
- Adhoc design decisions

The Need for Improving Machine Descriptions

The Problems:

- The specification mechanism for Machine descriptions is quite adhoc
  - Only syntax borrowed from LISP, neither semantics not spirit!
  - Non-composable rules
  - Mode and code iterator mechanisms are insufficient
- Adhoc design decisions
  - Honouring operand constraints delayed to global register allocation
    During GIMPLE to RTL translation, a lot of C code is required
  - Choice of insertion of NOPs

Symptom of Poor Specification Mechanism

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
  Size in terms of line counts for gcc-4.5.0

Target	*.md	*.c	*.h	Total
i386	35766	28643	15694	80103
mips	12930	12572	5105	30607

Symptom of Poor Specification Mechanism

- Machine descriptions are large, verbose, repetitive, and contain large chunks of C code
  Size in terms of line counts for gcc-4.5.0

Target	*.md	*.c	*.h	Total
i386	35766	28643	15694	80103
mips	12930	12572	5105	30607

- Machine descriptions are difficult to construct, understand, debug, and enhance

Typical Instruction Specification in GCC

```
(define_insn
    "movsi"
     [(set
       (match_operand:SI 0 "register_operand" "r")
       (match_operand:SI 1 "const_int_operand" "k")
       )]
    "" /*  C boolean expression, if required */
    "li %0, %1"
)
```


Typical Instruction Specification in GCC

Define instruction pattern

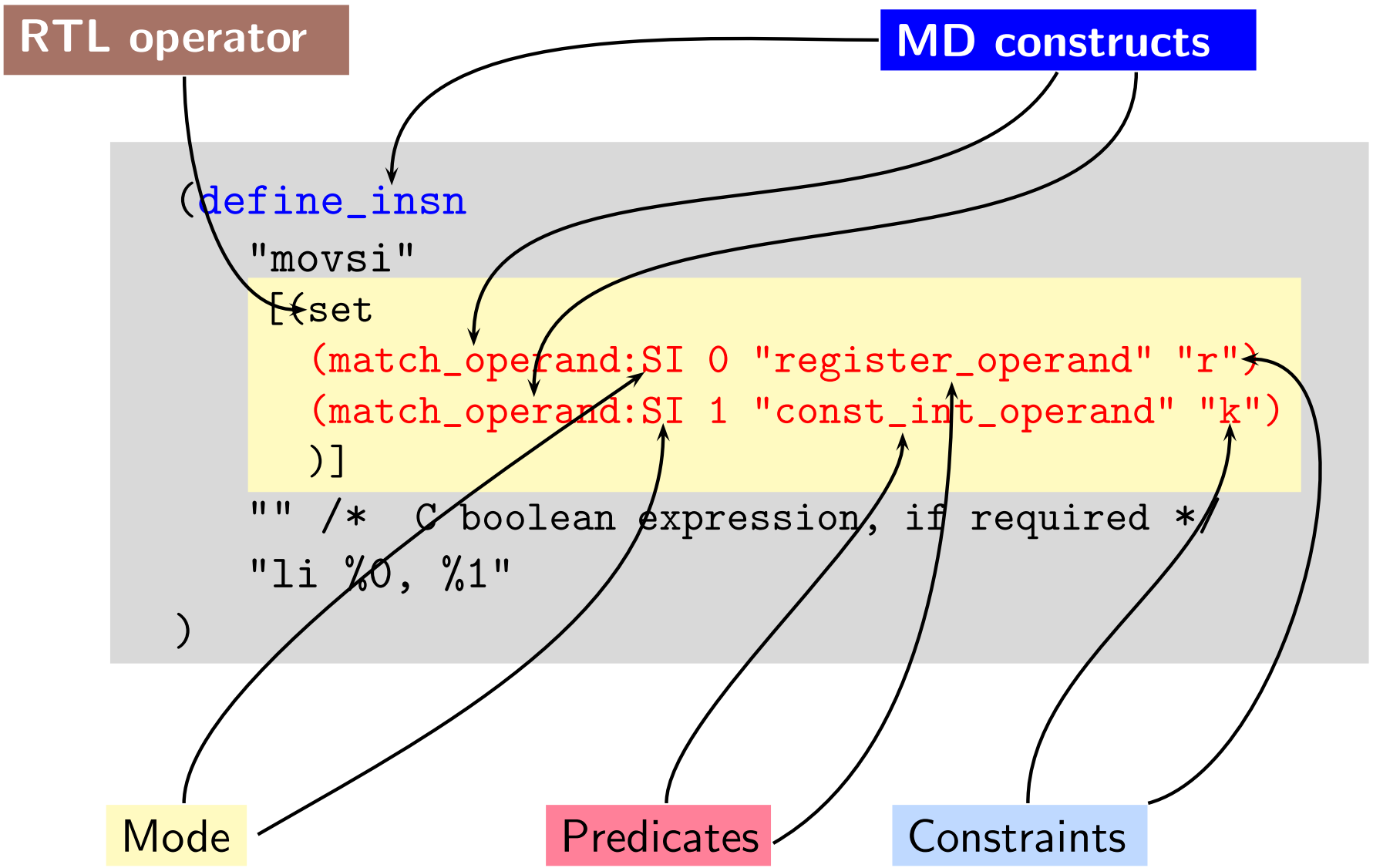
Standard Pattern Name

```
(define_insn
    "movsi"
     [(set
       (match_operand:SI 0 "register_operand" "r")
       (match_operand:SI 1 "const_int_operand" "k")
       )]
    "" /*  C boolean expression, if required */
    "li %0, %1"
)
```

RTL Expression (RTX):
Semantics of target instruction

target asm inst. =
Concrete syntax for RTX

Typical Instruction Specification in GCC



Design Flaws in Machine Descriptions

Multiple patterns with same structure

- Repetition of almost similar RTL expressions across multiple `define_insn` an `define_expand patterns`
  - Some Modes, Predicates, Constraints, Boolean Condition, or RTL Expression may differ everything else may be identical
  - One RTL expression may appears as a sub-expression of some other RTL expression
- Repetition of C code along with RTL expressions in these patterns.

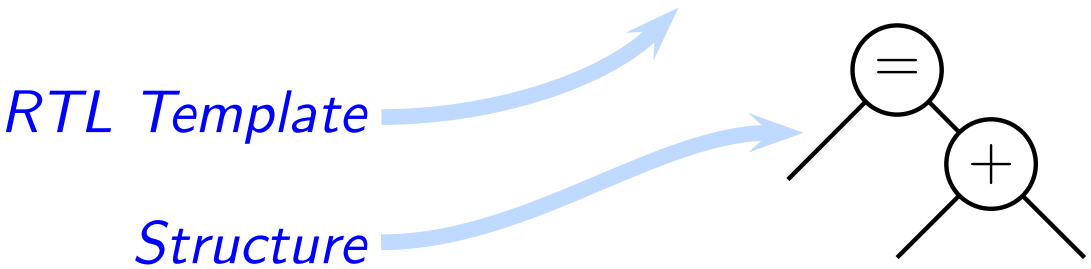
Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

RTL Template

Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

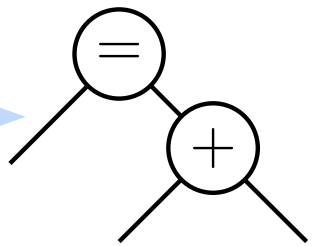


Redundancy in MIPS Machine Descriptions: Example 1

```
[(set (match_operand:m 0 "register_operand" "c0")
      (plus:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "p" "c2")))]
```

RTL Template

Structure



Details

Pattern name	*m*	*p*	*c0*	*c1*	*c2*
define_insn add<mode>3	ANYF	register_operand	=f	f	f
define_expand add<mode>3	GPR	arith_operand			
define_insn *add<mode>3	GPR	arith_operand	=d,d	d,d	d,Q

Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

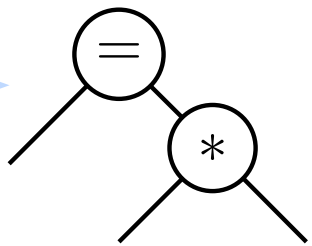
RTL Template

Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

RTL Template

Structure

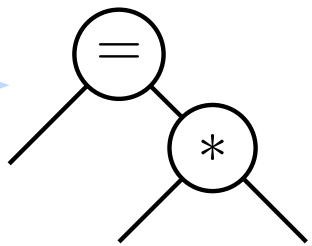


Redundancy in MIPS Machine Descriptions: Example 2

```
[(set (match_operand:m 0 "register_operand" "c0")
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
```

RTL Template

Structure



Details

Pattern name	*m*	*c0*	*c1*	*c2*
define_insn *mul<mode>3	SCALARF	=f	f	f
define_insn *mul<mode>3_r4300	SCALARF	=f	f	f
define_insn mulv2sf3	V2SF	=f	f	f
define_expand mul<mode>3	GPR			
define_insn mul<mode>3_mul3_loongson	GPR	=d	d	d
define_insn mul<mode>3_mul3	GPR	d,1	d,d	d,d

Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
      (match_operand:m 3 "register_operand" "c3")))]
```

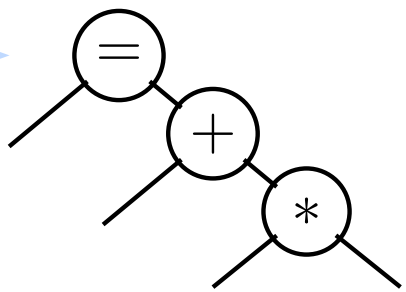
RTL Template

Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
      (mult:m (match_operand:m 1 "register_operand" "c1")
              (match_operand:m 2 "register_operand" "c2")))]
      (match_operand:m 3 "register_operand" "c3")))]
```

RTL Template

Structure

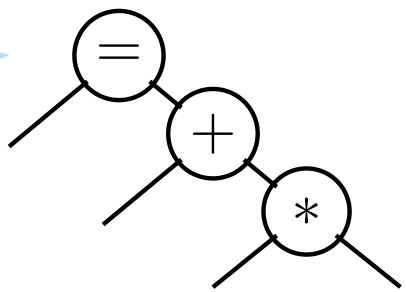


Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
     (mult:m (match_operand:m 1 "register_operand" "c1")
             (match_operand:m 2 "register_operand" "c2")))]
     (match_operand:m 3 "register_operand" "c3")))]
```

RTL Template

Structure



Details

Pattern name	m	c0	c1	c2	c3
*mul_acc_si	SI	=l*?*?,d?	d,d	d,d	0,d
*mul_acc_si_r3900	SI	=l*?*?,d*?,d?	d,d,d	d,d,d	0,1,d
*macc	SI	=l,d	d,d	d,d	0,1
*madd4<mode>	ANYF	=f	f	f	f
*madd3<mode>	ANYF	=f	f	f	0

Insufficient Iterator Mechanism

- Iterators cannot be used across define_insn, define_expand, define_peephole2 and other patterns
- Defining iterator attribute for each varying parameter becomes tedious
- For same set of modes and rtx codes, change in other fields of pattern makes use of iterators impossible
- Mode and code attributes cannot be defined for operator or operand number, name of the pattern etc.
- Patterns with different RTL template share attribute value vector for which iterators can not be used

Many Similar Patterns Cannot be Combined

```
(define_expand "iordi3"
   [(set (match_operand:DI 0 "nonimmediate_operand" "")
      (ior:DI (match_operand:DI 1 "nonimmediate_operand" "")
          (match_operand:DI 2 "x86_64_general_operand" "")))
   (clobber (reg:CC FLAGS_REG))]
   "TARGET_64BIT"
   "ix86_expand_binary_operator (IOR, DImode, operands); DONE;")

(define_insn "*iordi_1_rex64"
   [(set (match_operand:DI 0 "nonimmediate_operand" "=rm,r")
      (ior:DI (match_operand:DI 1 "nonimmediate_operand" "%0,0")
          (match_operand:DI 2 "x86_64_general_operand" "re,rme" )))
   (clobber (reg:CC FLAGS_REG))]
   "TARGET_64BIT
   && ix86_binary_operator_ok (IOR, DImode, operands)"
   "or{q}\t{%2, %0|%0, %2}"
   [(set_attr "type" "alu")
   (set_attr "mode" "DI")])
```

Measuring Redundancy in RTL Templates

MD File	Total number of patterns	Number of primitive trees	Number of times primitive trees are used to create composite trees
`i386.md`	1303	349	4308
`arm.md`	534	232	1369
`mips.md`	337	147	921

Part 3

Introduction to specRTL

Key Observation Behind specRTL

- Davidson Fraser insight

  Register transfers are target specific but their form is target independent

- GCC's approach
  - Use Target independent RTL for machine specification
  - Generate expander and recognizer by reading machine descriptions

  Main problems with GCC's Approach

  Although the shapes of RTL statements are target independent, they have to be provided in RTL templates

- Our key idea:

  Separate shapes of RTL statements from the target specific details

Specification Goals of specRTL

Support all of the following

- Separation of shapes from target specific details
- Creation of new shapes by composing shapes
- Associtiating concrete details with shapes
- Overriding concrete details

Software Engineering Goals of specRTL

- Allow non-disruptive migration for existing machine descriptions
  - Incremental changes
  - No need to change GCC source until we are sure of the new specification

  GCC must remain usable after each small change made in the machine descriptions

Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)

Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)
- *Abstract patterns* and *Concrete patterns*
  - Abstract patterns are shapes with "holes" in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information

Meeting the Specification Goals: Key Idea

- Separation of shapes from target specific details:
  - Shape ≡ tree structure of RTL templates
  - Details ≡ attributes of tree nodes
    (eg. modes, predicates, constraints etc.)
- *Abstract patterns* and *Concrete patterns*
  - Abstract patterns are shapes with "holes" in them that represent missing information
  - Concrete patterns are shapes in which all holes are plugged in using target specific information
- Abstract patterns capture *shapes* which can be concretized by providing details

Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: `extends`

Meeting the Specification Goals: Operations

- Creating new shapes by composing shapes: `extends`
- Associtiating concrete details with shapes: `instantiates`

Meeting the Specification Goals: Operations

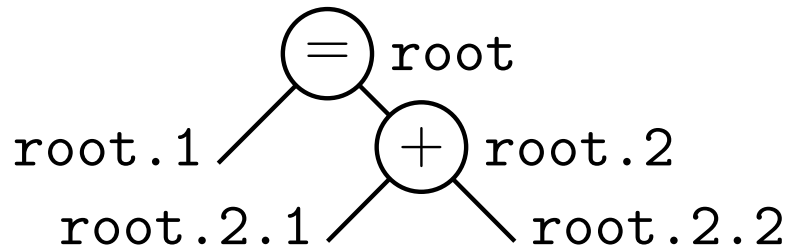
- Creating new shapes by composing shapes: `extends`
- Associtiating concrete details with shapes: `instantiates`
- Overriding concrete details: `overrides`

Properties of Operations

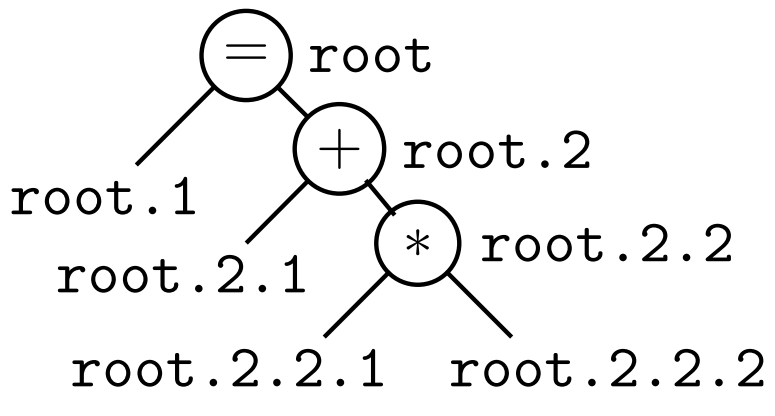
Operation	Base pattern	Derived pattern	Nodes influenced	Can change
`extends`	Abstract	Abstract	Leaf nodes	Structure
`instantiates`	Abstract	Concrete	All nodes	Attributes
`overrides`	Abstract	Abstract	Internal nodes	Attributes
	Concrete	Concrete	All nodes	Attributes

Creating Abstract Patterns

```
abstract set_plus extends set
{
  root.2 = plus;
}
```

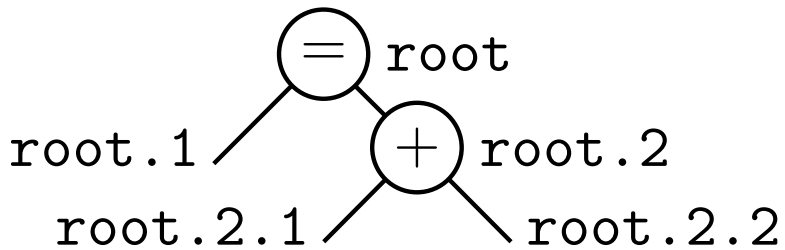



```
abstract set_macc extends
   set_plus
{
  root.2.2 = mult;
}
```



Creating Concrete Patterns

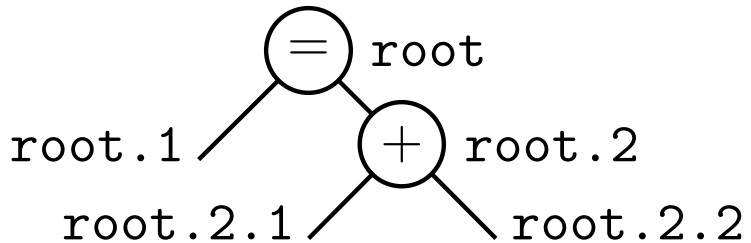
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
concrete add<mode>3.insn instantiates set_plus
{ set_plus(register_operand:ANYF:"=f",
           register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
concrete add<mode>3.expand instantiates set_plus
{ set_plus(register_operand:GPR:"",
           register_operand:GPR:"",
           arith_operand:GPR:"");
  root.2.mode = GPR;
}
```

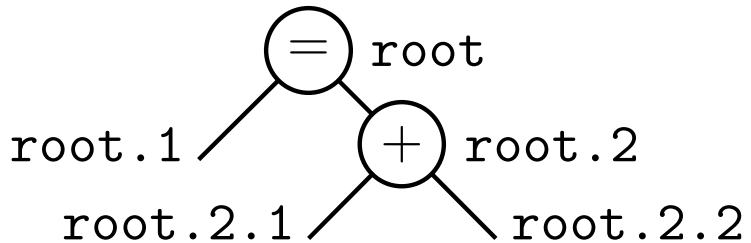

Generating Conventional Machine Descriptions

```
abstract set_plus extends set
{
  root.2 = plus;
}
```



Generating Conventional Machine Descriptions

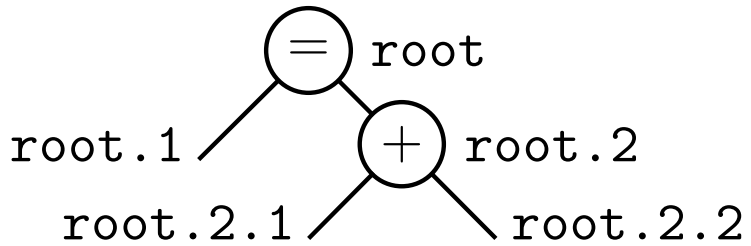
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
concrete add<mode>3.insn instantiates set_plus
{ set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
{: /* Conventional Machine Description Fragments */ :}
```

Generating Conventional Machine Descriptions

```
abstract set_plus extends set
{
  root.2 = plus;
}
```



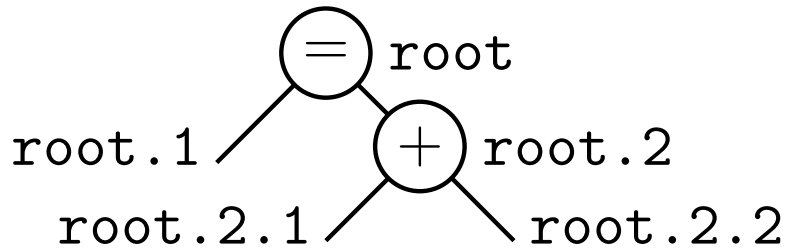

```
concrete add<mode>3.insn instantiates set_plus
{ set_plus(register_operand:ANYF:"=f", register_operand:ANYF:"f",
           register_operand:ANYF:"f");
  root.2.mode = ANYF;
}
{: /* Conventional Machine Description Fragments */ :}
```

Resulting MD Specification

```
(define_insn "add<mode>3"
[(set (match_operand:ANYF 0 "register_operand" "=f")
      (plus:ANYF (match_operand:ANYF 1 "register_operand" "f")
                 (match_operand:ANYF 2 "register_operand" "f")))]
/* Conventional Machine Description Fragments */
)
```


Overriding Details

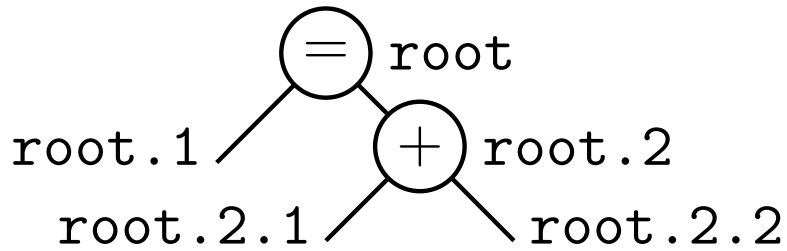
```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
concrete add<mode>3.expand instantiates set_plus
{ set_plus(register_operand:GPR:"",
           register_operand:GPR:"",
           arith_operand:GPR:"");
  root.2.mode = GPR;
}
```

Overriding Details

```
abstract set_plus extends set
{
  root.2 = plus;
}
```




```
concrete add<mode>3.expand instantiates set_plus
{ set_plus(register_operand:GPR:"",
           register_operand:GPR:"",
           arith_operand:GPR:"");
  root.2.mode = GPR;
}
```

```
concrete *add<mode>3.insn overrides  add<mode>3.expand
{ allconstraints = ("=d,d", "d,d", "d,Q"); }
```


Some More Examples

Omitting conventional MD fragments

```
concrete *mul<mode>3.insn instantiates set_mult
{ set_mult(register_operand:SCALARF:"=f",
           register_operand:SCALARF:"f",
           register_operand:SCALARF:"f");
  root.2.mode = SCALARF;
}
```


Some More Examples

Omitting conventional MD fragments

```
concrete *mul<mode>3.insn instantiates set_mult
{ set_mult(register_operand:SCALARF:"=f",
           register_operand:SCALARF:"f",
           register_operand:SCALARF:"f");
  root.2.mode = SCALARF;
}
```

```
concrete *mul<mode>3_r4300.insn overrides *mul<mode>3.insn
{}
concrete mulv2sf3 overrides *mul<mode>3.insn
{ SCALARF -> V2SF; }
```


Part 4

Conclusions

Current Status and Plans for Future Work

- specRTL parser has been augmented with semantic checks
  Emitting conventional machine descriptions is pending
- i386 move instructions and spim add instructions have been rewritten
  Other instructions are being rewritten
- Suggestions have been received to improve the syntax

Conclusions

- Separating shapes from concrete details is very helpful
- It may be possible to identify a large number of common shapes
- Machine descriptions may become much smaller
  Only the concrete details need to be specified
- Non-disruptive and incremental migration to new machine descriptions
- GCC source need not change until these machine descriptions have been found useful

Last but not the least ...

Thank You!